Optimised deployment of critical applications in Infrastructure-as-a-Service clouds*

Imre Kocsis, Zoltán Ádám Mann, Dávid Zilahi
Budapest University of Technology and Economics

Abstract

In this paper, we extend the classic data center allocation optimization problem for critical tenant applications that need guarantees on the required resource capacities. We identify a set of representative, user-issuable constraints and new optimization objectives and establish a mathematical and corresponding integer programming formulation. Using a typical Network Function Virtualization application as a case study, we show the viability of the approach and present an initial scalability assessment. Our results demonstrate the advantage of combining the conflicting objectives of the tenants and the provider in a single optimization problem.

Keywords: data center allocation optimization; virtual machine placement; Infrastructure as a Service; integer linear programming; Network Function Virtualization; deployment of critical applications

1 Introduction

Contemporary general-purpose cloud services provide a compelling and proven option for a variety of workloads, ranging from web page serving to Big Data analytics. Resource pooling is an essential characteristic of these services, "with different physical and virtual resources dynamically assigned and reassigned on-demand" [33]. To drive down operational costs, operators optimize this mapping continuously and independently from the tenants, by consolidating workload to the minimal number of hosts and taking unused resources offline. In the Infrastructure as a Service (IaaS) domain, this activity is called *allocation optimization*, referring to the allocation of Virtual Machines (VMs) to hypervisors running them.

Aggressive allocation optimization is made possible by cloud service properties that tenants have learned to accept:

- Usually, the mapping of tenant applications to resources is hidden from the tenants – they cannot even observe, let alone control it
- Extra-functional service parameters are specified only roughly, if at all
- Service Level Agreements (SLAs) are very permissive towards providers

In this setting, it is largely taken as a fact of life that the extra-functional properties of the service, like performance, can have significant *variability* [29]. Variability here means instability as well as heterogeneity. Focusing on IaaS performance, this means that virtual resource performance does not remain the same with the progress of time or across the instances – even if it is specified by the provider using some rough measure of capacity (e.g., the "EC2 Compute Unit" of Amazon Web Services).

The fundamental reason for performance instability – and to an extent, heterogeneity – is simply tenants sharing resources. Just as processes compete for the finite CPU time in an operating system (and also influence each other indirectly, e.g. through caches), VMs on the same hypervisor can have an impact on each other's performance. Host, operating system and network virtualization technologies do provide a range of mechanisms for the performance isolation of tenants – colloquially, for avoiding the "noisy neighbor" effect. These facilities vary in effectiveness: e.g., CPU usage caps tend to be an effective way for the CPU performance isolation of VMs, but for number-crunching applications, cache and memory bandwidth may become a bottleneck [25]. Timeliness-critical applications may require dedicated resources (e.g. dedicated CPU cores) for stable performance. However, even though these means are available to the provider, for the tenants performance isolation is also a not observable and not controllable aspect of cloud services. The same holds for other runtime extra-functional assurance mechanisms (e.g. runtime fault tolerance techniques).

*This paper was published in *International Journal of Cloud Computing*, 6(4):342-362, 2017. DOI: 10.1504/IJCC.2017.10011288

With the increasing push for the "cloudification" of timeliness-critical and very high availability applications, this status quo is rapidly changing. For true application-level performance and dependability guarantees for such systems, tenants have to become able to formulate various constraints on the way operators fulfill their (virtual) resource requirements. These constraints must be carefully chosen, so that they indeed allow tenants to have sufficient control but on the other hand also leave providers enough opportunities for effective allocation optimization. Examples for such constraints may include the need for dedicated physical host subsystems for a VM or affinity and antiaffinity rules on VMs or VM types. Our main goal in this paper is to give a – to our knowledge first – treatment to the problem of IaaS operator allocation optimization in this novel setting.

1.1 Allocation optimization in general-purpose clouds

IaaS offers tenants virtualized resources; most importantly, VMs that can be created and destroyed on demand. Rapid elasticity of the resources and usage-based billing enable tenants to continuously adapt their resource usage to the dynamically changing needs. This way, IaaS can significantly lower overall operational costs for tenants, even if the "unit cost" of a cloud resource is higher than for own resources. From the provider point of view, IaaS can be provided efficiently because sharing a large pool of resources between largely independent tenants "statistically multiplexes" the varying loads [42] – in addition to simple economies of scale. However, as a rule, tenants have only limited and indirect influence over the deployment of their reservations onto physical resources, making critical applications vulnerable to a variety of faults, e.g., common mode hardware and capacity faults [26]. This holds not only for today's general-purpose public IaaS clouds, but mostly for the leading open IaaS software frameworks, too.

Operators and tenants strive to optimize their operations with conflicting goals. Operators aim at consolidating tenant VMs to as few hypervisors as possible to save power and cooling costs by switching off unused hosts [30]. Reliability, availability, performance stability and homogeneity are secondary concerns [23]; if needed, tenants must equip their applications with the appropriate resiliency mechanisms at the application level. Applying these mechanisms – e.g., maintaining online spare capacity – has serious cost impacts. Thus, redundancy has to be dynamically managed as well, taking into account variable application workload as well as variable performability [3].

1.2 Emerging user requirements for critical applications

A number of emerging cloud-delivered services have to adhere to strict constraints on various Service Level Objectives (SLOs). A prime example is Network Function Virtualization (NFV) [15]: the current push in the telco domain to migrate network functions from dedicated appliances to IaaS. The first step of this evolutionary process is that telco providers migrate existing legacy appliances largely without modification into sets of VMs, weaved together either by standard network virtualization or increasingly by Software Defined Networks (SDNs). These so-called Virtualized Network Functions (VNFs) provide telco services ranging from simple IP packet stream manipulation to sophisticated IP-Multimedia Subsystem specifications. Crucially, they have to provide *telco-grade* services; that is, the allowed delay, response time as well as availability thresholds are very stringent [12].

In order to adhere to these requirements at the service level, tenants need the enforcement of analogously stringent extra-functional requirements on the underlying IaaS services. Ideally, tenants should be able to submit these requirements to the operator in the form of constraints formulated on established resource service attributes (and the metrics of these). An example would be that the variability of single-core processing capacity for a VM (measured e.g. as the coefficient of variation of the amount of available CPU cycles for single-threaded execution, summed for a small sliding window) should be lower than a predefined value. The operator should then be able to make the appropriate deployment decisions, including the configuration of appropriate isolation mechanisms, based on this declarative request.

There are two major problems with this idealized approach. First, there is not even consensus on the attributes (and metrics) for characterizing NFVIaaS resource services. The NFV Service Quality Metrics specification [14] establishes the Speed, Accuracy and Reliability categories and declares the most important attributes and metrics for VMs and virtual networks (VNs) that take part in delivering VNFs; however, the specification is new and thus application experience is still lacking. Also, [14] covers the set of known possible cloud service impairments incompletely; it focuses on the "pain points" that are deemed the most important from the point of view of existing NFV applications. Other cloud service quality attribute/metric and KPI taxonomies are even less appropriate for this purpose; e.g. the Service Measurement Index (SMI) [11] emphasizes business-related service properties at the expense of expressing runtime technical compliance. Second, deriving allocation and configuration decisions from constraints on metrics has been and remains to be a nontrivial exercise; in many cases, it is theoretically possible, but practically infeasible. Instead, a practical approach is to use a mix of resource service quality constraints

and direct, user-issued, explicit allocation and configuration directives. The various architectural guidance and interface specifications published by the European Telecommunications Standards Institute (ETSI) for NFV follow this pattern (see [13]).

It is still very early to try to set up a definitive taxonomy of these. However, the applicable parts of the ETSI NFVIaaS specification can be interpreted as requirements for enabling users to ask for capabilities that *already exist* in the hypervisor domain – just not visible for and requestable by the tenants in current cloud environments. In line with the NFV specifications, our prior experience in virtualization design for QoS and our current work with a telco NFV application (see later) show that a tenant may want to request at least the following deployment properties:

1. With respect to the partitionable resources of the hypervisor hosts, *capacity guarantees* for the VMs that are stable even at fine time scales. At the current state of technology, CPU cycles, I/O bandwidth (with possibly network-attached storage), network packet or data rate and physical memory space fall under this category. Modern hypervisors can be configured to provide allowance guarantees on these capacities. As the guarantees are realized through scheduling, the abstraction of resources "dedicated" this way is not necessarily perfect; however, at least for CPU scheduling even real-time hypervisors are emerging (for various levels of hardness).

2. Some applications are sensitive to contention of the today non-partitionable resources, like CPU caches and various host-internal bandwidths (memory, device buses, interconnects). To protect these applications, a tenant may want to ask for *dedicated* physical resources. In virtualization, what is mostly available today is dedicating *CPU cores* or sets of cores to VMs (at least for SMP systems). Another requirement can be *pinning* VMs to cores to avoid the performance impact of migration between cores or CPUs. VMs may also want dedicated access to physical devices as network interface cards and local storage, but the reason for this may be increasing performance instead of ensuring performance capacity guarantees.

3. From the point of view of deployment to hypervisors, the requirements posed by most runtime dependability techniques can be formulated as VM-VM *antiaffinity* and *affinity* rules. VMs forming a fault tolerant cluster are usually required to be spread out across physical hosts to protect against hypervisor host single faults. Conversely, for realizing the watchdog pattern the optimal solution may be to run the worker and watchdog VMs on the same host. Affinity rules can also be important for application performance, since communication between VMs on the same host can be much faster than across the network.

4. A *proscription of live VM migration* may be necessary for avoiding the short, but detectable VM stalls that can accompany moving a live VM from a host to another.

In IaaS for critical applications (and specifically NFV), the two most fundamental resource services are Virtual Machines and Virtual Networks. We formulated the above deployment requirement categories for VMs – although VNs have just as nontrivial deployment aspects. We chose to focus on the open problem of VM deployment optimization for critical applications in this work, restricting the treatment to physical setups where

- host-to-host network connections do not radically differ in physical latencies (or the difference does not matter for the applications) and
- the bandwidth bottleneck may be the host network interface cards, but not the networking fabric.

These assumptions may not hold at the scale of large data centers, but are typically true for single racks or connected local groups of racks. This is already in the size domain where current commercial data center "cloud units" fall for NFV; also, many critical distributed applications retain this level of locality with data center level and geographic dispersion of whole clusters handled as a separate problem. This way, our results approach an open problem with directly applicable results and hopefully provide useful insights for the broader (and mathematically even more challenging) problem of integrated VM and VN deployment optimization.

2 Previous work

The optimization problem of how to best allocate VMs to the available physical machines (PMs) has received a lot of attention in the last couple of years, leading to a wide variety of proposed problem models and algorithms.

In several works, only the CPU load of VMs and the CPU capacity of PMs is considered [5, 17, 19, 27, 10]. Considering also other resource dimensions like memory [16, 4], or even memory and disk and/or network I/O [7, 22, 37, 40] makes the problem somewhat more complicated but also more useful in practice. The objective of

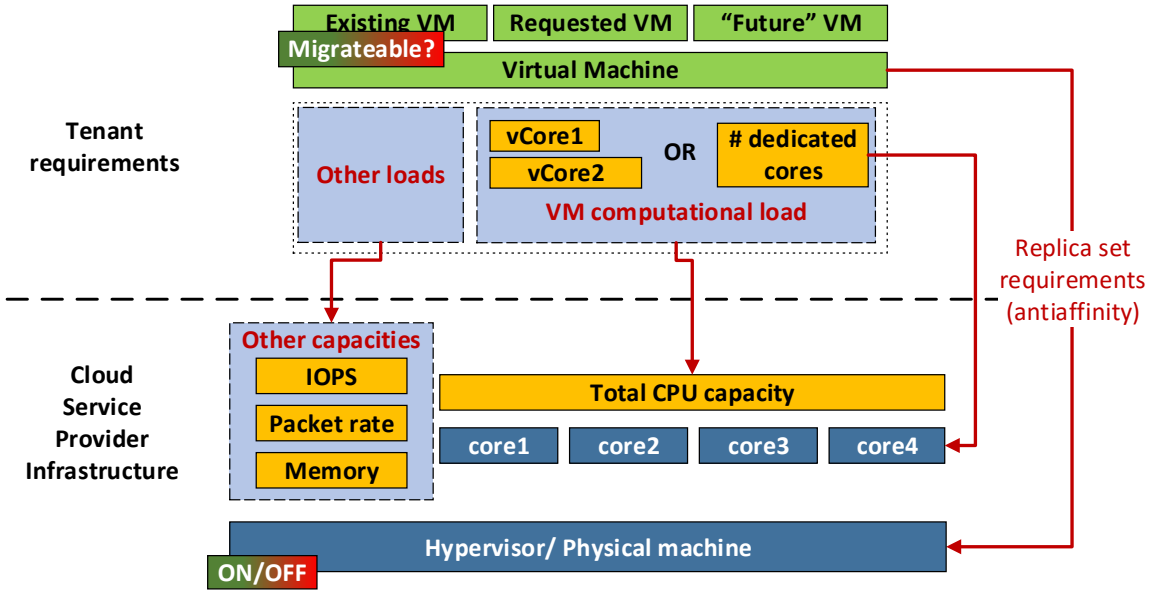

Figure 1: Overview of the allocation and deployment modeling approach

the optimization is in many cases the number of active PMs [9, 36, 38, 39], often together with other metrics like the number of SLO violations [5, 8, 41, 43]. More details on the used problem models can be found in [32].

Concerning the algorithmic techniques suggested so far in the literature, the situation is a bit less heterogeneous. Many researchers proposed to take advantage of the similarity between the VM allocation problem and the well-known bin-packing problem and to adapt packing heuristics like First-Fit and Best-Fit to VM allocation [6, 8, 24, 40, 41]. However, VM allocation is actually a much harder problem than bin packing [31], limiting the applicability of these heuristics to the simplest versions of the VM allocation problem. Other researchers proposed to use metaheuristics like genetic algorithms and ant colony optimization [16, 18] or sophisticated proprietary heuristics [2]. There were also some attempts to devise exact algorithms for VM allocation, using mostly mixed integer linear programming and related techniques [20, 28].

3 Proposed approach

Our paper extends classic IaaS allocation optimization problem models [30] with VM placement for critical applications. For this purpose, we extend the set of *constraints* that deployments have to adhere to and modify the optimization *objectives* to reflect both tenant and provider goals.

Figure 1 gives an overview of our allocation and deployment modeling approach. The host machines of the operator have various capacities: storage IOPS, network packet rate, physical memory, and CPU capacity. These are all partitionable capacities, with broad hypervisor support for partitioning them; notably, memory bandwidth is not present (although it can be very important for critical applications). The amount of accessible storage is not modeled as it is usually not a concern in the given domain. In addition to total CPU capacity, individual physical cores are also present as first-class concepts. The hosts can be switched on or off.

Tenant VMs can be already existing or ones that have just been requested and wait for being provisioned. "Future" VMs are not requested currently, but a tenant has declared that they will be needed in the future and the provider may be asked for assurances on being able to provide them later in a timely manner. For some VMs, the tenants may decide to prohibit migration (limiting the rearrangement of VM allocation in data center optimization).

All VMs have computational load and "other resource load" requirements. We focus on computational capacity, as it is a representative problem for the similar handling of the other capacity types and has an emphasis in NFV currently. While the "other load" requirements are expressed as single scalars on a per capacity type basis, computational capacity requirements are either given as the load on a per CPU core basis, or as the number of dedicated cores requested. The load of a VM can be tagged as capacity that must be guaranteed.

In order to enable cluster and application dependability, a tenant can request that at most k VMs of a set of VMs may be deployed to the same host. In classic dependability terminology, this is a *replica set* allocation constraint, but this facility can be used to express a wide range of *antiaffinity* rules. *Affinity* rules could be introduced similarly;

we chose to omit these to keep the model simpler and because antiaffinity rules are far more important for critical applications.

The objective function in the majority of the works for general-purpose cloud data center allocation optimization is energy consumption, captured through the number of switched-on hosts. We introduce two additional aspects: the amount of future requests that can be fulfilled without bringing further hypervisors online (a usually lengthy operation) and the impact of single-host failures. The former is modeled by the number of "future" VMs that could be allocated, while the latter is expressed as the highest number of VMs any tenant can lose due to a host failure. To highlight the trade-off between these goals, we chose to build a single weighted composite objective function from them. Our approach towards modeling optimality is intended to be a representative example; as it is common with multi-aspect optimization, what constitutes an "optimal" solution is heavily problem-dependent.

In the following, we present a mathematical model and the corresponding integer programming formulation fitting this approach, followed by a case study.

4 Problem formalization

The set of available PMs is denoted by P. PM $p \in P$ has $pcn(p) \in \mathbb{Z}^+$ cores (pCPUs) and $pcc(p) \in \mathbb{R}^+$ computational capacity per core. The set of cores of PM p is denoted by $pC(p)$; further, let

$$PC = \bigcup_{p \in P} pC(p)$$

denote the set of all PM cores. Beyond the CPU, the capacity in further resource dimensions is given by $cap(p, r) \in \mathbb{R}^+$ for $p \in P$ and $r \in R = \{memory, ingressPacketRate, egressPacketRate, IOPS\}$.

The set of VMs, V, is comprised of three subsets: $V = V_0 \cup V_1 \cup V_2$, where V_0, V_1, and V_2 are pairwise disjoint. V_0 contains the VMs that are already accommodated by PMs, whereas V_1 consists of newly requested VMs that need to be allocated now. V_2 contains reservations for future VMs that need not be allocated right now; they just represent indications for future workload.

For each VM $v \in V$, the number of its processor cores (vCPUs) is denoted by $vcn(v) \in \mathbb{Z}^+$, the computational capacity per core requested for the VM is denoted by $vcc(v) \in \mathbb{R}^+$, and the set of its cores is denoted by $vC(v)$. The set of all VM cores is denoted by

$$VC = \bigcup_{v \in V} vC(v).$$

The load is given for all VMs and each resource dimension:

- If the customer requested guaranteed capacity for the given VM and the given resource dimension, then this value is used as the VM's load.
- Otherwise, in the case of an existing VM ($v \in V_0$), either its current load can be used, or an estimate for the near future based on past observations; however, if this value is higher than the requested capacity, then the requested capacity is used.
- Otherwise, a given percentage of the requested capacity is used, taking into account that users typically overestimate the necessary capacity for their VMs [35].

In any case, for a VM $v \in V$ and for its core $vc \in vC(v)$, $vcl(v, vc) \in \mathbb{R}^+$ denotes the computational load of that VM core. For VM $v \in V$ and any further resource type $r \in R$, $vload(v, r) \in \mathbb{R}^+$ is its load in this resource dimension.

There is a current mapping of VMs to PMs, represented by a function $map_0 : V_0 \to P$. Our goal is to compute a new mapping $map : V' \to P$, where $V_0 \cup V_1 \subseteq V' \subseteq V$, i.e., all VMs in $V_0 \cup V_1$ must be mapped to PMs and possibly also some in V_2.

The cost of a mapping is comprised of three terms. The first term is the number of active PMs. A PM is called active if at least one VM is mapped to it. PMs that are not active can be switched to sleep mode, thus considerably reducing their energy consumption. Therefore, to save energy, we should minimize the number of active PMs, denoted by $A(map)$ and calculated as

$$A(map) = \big|\, \{p \in P : \exists v \in V \; map(v) = p\} \,\big|.$$

The second term of the cost of a mapping rewards the allocation of VMs in V_2 to PMs. The number of VMs in V_2 that are allocated to PMs is given by $B(map) = |V' \setminus (V_0 \cup V_1)|$.

The third term of the cost of a mapping penalizes the impact of the failure of a PM on a tenant. To make this more precise, let T denote the set of tenants, and for each $t \in T$, let $VT(t) \subseteq V$ denote the set of VMs belonging to this tenant. (Therefore, the subsets $VT(t), t \in T$ form a partition of V.) Let $C(map)$ denote the highest number of VMs of the same tenant allocated to the same PM. Clearly, this represents the worst-case impact of a PM failure on a tenant in terms of the number of affected VMs, and can be calculated as

$$C(map) = \max \Big\{ \big| \{v \in VT(t) : map(v) = p\} \big| : t \in T, p \in P \Big\}.$$

The overall objective function that we want to minimize is

$$\alpha \cdot A(map) - \beta \cdot B(map) + \gamma \cdot C(map),$$

where α, β, and γ are given non-negative constants. It should be noted that $A(map)$ and $C(map)$ are to be minimized, whereas $B(map)$ is to be maximized.

The solution, encoded by map, must satisfy the following capacity constraints:

$$\forall p \in P, r \in R : \sum_{v \in V : map(v) = p} vload(v, r) \leq cap(p, r) \tag{1}$$

These relate to all resource types other than the CPU. For the CPU, the situation is more complex because some VMs may require dedicated cores. Let $V_{ded} \subseteq V$ denote the set of VMs for which dedicated CPU cores were requested and $V_{nonded} = V \setminus V_{ded}$ the rest. For a PM $p \in P$, let $V_{ded}(p, map) = \{v \in V_{ded} : map(v) = p\}$ denote the set of VMs that need dedicated cores and are mapped to p, and $V_{nonded}(p, map) = \{v \in V_{nonded} : map(v) = p\}$ the set of VMs that do not need dedicated cores and are mapped to p. The VMs mapped to PM $p \in P$ can be scheduled on p's CPU if and only if all of the following constraints are fulfilled:

$$\forall v \in V_{ded}(p, map) \cup V_{nonded}(p, map) : \ vcc(v) \leq pcc(p), \tag{2}$$

$$\sum_{v \in V_{ded}(p, map)} vcn(v) \leq pcn(p), \tag{3}$$

and

$$\sum_{v \in V_{nonded}(p, map)} \sum_{vc \in vC(v)} vcl(v, vc) \leq \left(pcn(p) - \sum_{v \in V_{ded}(p, map)} vcn(v) \right) \cdot pcc(p). \tag{4}$$

Constraint (2) ensures that each VM is mapped to a PM with cores of sufficient capacity. Constraint (3) ensures that the number of cores of PM p is sufficient for the VMs mapped to it that require dedicated cores, whereas constraint (4) ensures that the cores of p that remain for the VMs not requiring dedicated cores have sufficient total capacity.

The further special constraints for critical VMs can be formulated as follows:

- VM v must not be migrated
  - If $v \in V_0$:

$$map(v) = map_0(v) \tag{5}$$

  - If $v \in V_1$, then v needs to be placed in such a way that it will never lead to an SLA violation, because the placement of v cannot be changed anymore. In principle, a problem could arise if multiple non-migrateable VMs that are mapped to the same PM start to use heavily the same resource, because such overuse could not be remedied by migration. However, our approach ensures that VMs requiring guaranteed capacity (and critical VMs will typically belong to this category) do indeed obtain that capacity (since we use the required capacity as load for such VMs during our allocation decisions, even if their actual load is lower), so that no SLA violation can occur for such VMs.
- For a set of VMs $\overline{V} \subseteq V$, at most k of them can be allocated to the same PM:

$$\forall p \in P : \big| \{v \in \overline{V} : map(v) = p\} \big| \leq k \tag{6}$$

5 Integer Linear Programming model

Let $n = |V|$ and $m = |P|$. VMs are indexed as v_i ($i = 1, \ldots, n$), PMs are indexed as p_j ($j = 1, \ldots, m$). In order to formulate the above problem as an integer linear program (ILP), the following binary variables are introduced:

$$Alloc_{i,j} = \begin{cases} 1 & \text{if } v_i \text{ is allocated on } p_j \\ 0 & \text{otherwise} \end{cases}$$

$$Active_j = \begin{cases} 1 & \text{if } p_j \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$

In addition, there is an integer variable C, corresponding to $C(map)$ above.

Using these variables, the integer program can be formulated as follows (if not stated otherwise, $i = 1, \ldots, n$ and $j = 1, \ldots, m$):

$$\text{minimize} \quad \alpha \cdot \sum_{j=1}^{m} Active_j - \beta \cdot \sum_{v_i \in V_2} \sum_{j=1}^{m} Alloc_{i,j} + \gamma \cdot C \tag{7}$$

$$\text{subject to} \quad \sum_{j=1}^{m} Alloc_{i,j} = 1, \qquad \forall v_i \in V_0 \cup V_1 \tag{8}$$

$$\sum_{j=1}^{m} Alloc_{i,j} \leq 1, \qquad \forall v_i \in V_2 \tag{9}$$

$$Alloc_{i,j} \leq Active_j, \qquad \forall i,\ \forall j \tag{10}$$

$$\sum_{v_i \in VT(t)} Alloc_{i,j} \leq C, \qquad \forall t \in T,\ \forall j \tag{11}$$

$$\sum_{i=1}^{n} vload(v_i, r) \cdot Alloc_{i,j} \leq cap(p_j, r), \qquad \forall j,\ \forall r \in R \tag{12}$$

$$vcc(v_i) \cdot Alloc_{i,j} \leq pcc(p_j), \qquad \forall i,\ \forall j \tag{13}$$

$$\sum_{v_i \in V_{ded}} vcn(v_i) \cdot Alloc_{i,j} \leq pcn(p_j), \qquad \forall j \tag{14}$$

$$\sum_{i=1}^{n} cpu\_load(i,j) \cdot Alloc_{i,j} \leq pcn(p_j) \cdot pcc(p_j), \qquad \forall j \tag{15}$$

$$Alloc_{i,j} = 1, \quad \text{if } v_i \in V_0 \text{ must not be migrated and } map_0(v_i) = p_j \tag{16}$$

$$\sum_{v_i \in \overline{V}} Alloc_{i,j} \leq k, \quad \text{if at most } k \text{ of } \overline{V} \text{ can be on the same PM}, \quad \forall j \tag{17}$$

$$Alloc_{i,j} \in \{0,1\}, Active_j \in \{0,1\}, \qquad \forall i,\ \forall j \tag{18}$$

The objective function (7) is the same as before, consisting of the number of active PMs, the number of VMs of V_2 that could be allocated, and the maximum number of VMs of the same tenant on the same PM. Equation (8) ensures that each VM in $V_0 \cup V_1$ is allocated to exactly one PM, whereas Equation (9) ensures that each VM in V_2 is allocated to at most one PM. Constraint (10) ensures that for a PM p_j to which at least one VM is allocated, $Active_j = 1$. Together with the objective function, this ensures that $Active_j = 1$ holds for *exactly* those PMs that accommodate at least one VM. Using similar logic, constraint (11) ensures that the value of C is at least $C(map)$; together with the objective function, it is guaranteed that C will have exactly this value.

Constraints (12)-(17) are mostly straight-forward formulations of constraints (1)-(6) in terms of the binary variables $Alloc_{i,j}$. The quantity $cpu\_load(i,j)$ in (15) is the CPU capacity that v_i would use up if allocated on p_j, calculated as follows:

$$cpu\_load(i,j) = \begin{cases} vcn(v_i) \cdot pcc(p_j) & \text{if } v_i \in V_{ded} \\ \sum_{vc \in vC(v_i)} vcl(v_i, vc) & \text{if } v_i \in V_{nonded} \end{cases}$$

If $v_i \in V_{nonded}$, then this is the total CPU load of v_i, not depending on j. If $v_i \in V_{ded}$, then it may be higher than the total CPU load of v_i and it also depends on j. However, from an integer programming point of view, the most important is that $cpu\_load(i,j)$ is a constant for each i, j.

6 Case study

To demonstrate our approach, we model the deployment of a specific, nontrivial NFV application, solve the problem for various weight values in the cost function and interpret the results. We use Gurobi [21] as the solver in our current toolchain. A preliminary scalability analysis is provided in section 6.4.

6.1 Project Clearwater

According to the website of the Clearwater Project (`http://www.projectclearwater.org`), "Clearwater is an open source implementation of IMS (the IP Multimedia Subsystem) designed from the ground up for massively scalable deployment in the Cloud to provide voice, video and messaging services to millions of users." At its core, Clearwater is a standards-based (VoIP) telephony "switching center" that is quickly becoming a standard example in NFV research. (The plethora of more sophisticated services that an IMS service has to provide is irrelevant from the point of view of this work.) The big telco equipment providers all have their IMS implementations; however, Clearwater is open source and explicitly engineered to be deployable in NFV IaaS environments as a so-called Virtual Network Function. It implements the IMS standards in such a way that the various functional components defined by the IMS standards directly map to its component services. It is possible to deploy all Clearwater components into a single VM or onto a single physical host; however, it is mainly intended to be deployed in a distributed way, by placing component service instances into separate VMs. Figure 2 demonstrates the topology of typical distributed Clearwater deployments; nearly all components are scalable as clusters of VMs running the same component code.

For reference, we give a short description of the functionality of the components. For more details, the interested reader is kindly referred to the IMS standard [1] and the documentation of the Clearwater project [34].

Bono implements the P-CSCF IMS function – acting as an edge proxy between users and the SIP routers.

Sprout SIP router, which is responsible for most of the I-CSCF and S-CSCF functions. It handles client authentication and routes SIP requests – e.g. call setup and teardown message flows – between users (and application servers). Sprout also features a built-in MMTel application server to enable multimedia communication.

Homestead is a Home Subscriber Service (HSS), the IMS standard facility for storing subscriber data. Homestead can serve as a mirror, but it can also serve as a master store.

Homer serves as a data store (XDMS) for the MMTel application server.

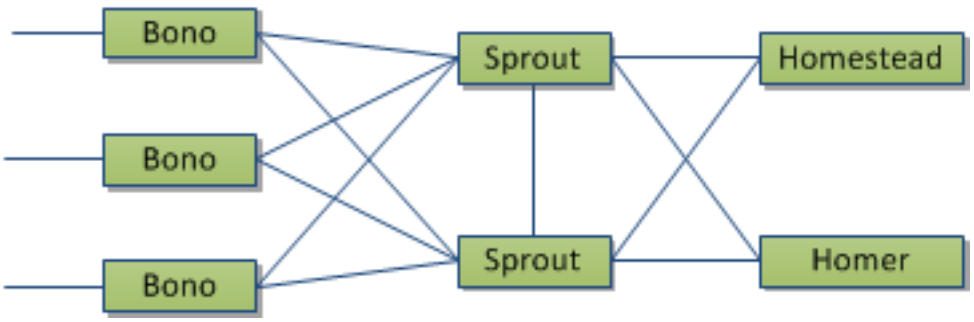

Figure 2: VMs and dependencies in a typical Clearwater deployment

6.2 An initial deployment

We begin our case study with a hypothetical (but realistic) Clearwater deployment, for which the allocation has been performed manually. Available in the data center are two "big" and two "small" PMs; their capacities are listed in Table 1. Figure 3 visualizes how the CPU, Memory and IOPS capacities are used by a Homer, a Homestead, two Bono and two Sprout instances, and two VMs belonging to another tenant. For the numerical data, see Table 2. (Note that Sprout instances have two cores, whereas the other Clearwater components have a single core. The two VMs from another tenant have two and four cores, respectively.) To keep the discussion simpler, we omitted the network (rate) capacity because it is not a major concern in the deployments that we have hands-on experience with.

PM type	Cores	Core capacity (GHz, HT enabled)	Memory (GB)	IOPS
bigPM	6	8	32	10000
smallPM	4	6	8	350

Table 1: Parameters of the available physical machine types

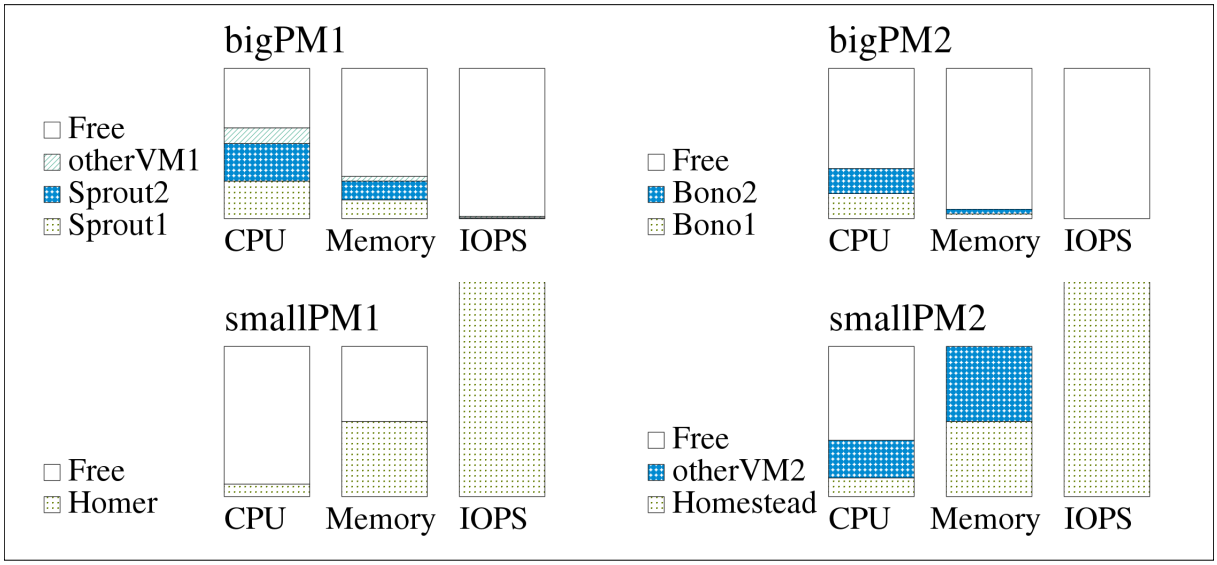

Figure 3: The original deployment

It is clear that this deployment is suboptimal in multiple aspects. The I/O capabilities of the two smaller physical machines (smallPM1, smallPM2) are saturated, while the utilization of the two bigger machines is low. Also, smallPM2 is very low on free memory. Overall, this deployment is inefficient in the sense that some resources are saturated, while there are also significant unused resources. With a better deployment, we might be able to turn off one or two machines. Another aspect concerns fault tolerance. In general clouds, it is usually not possible to set physical redundancy rules within one zone. This is how it can happen that VMs of the same cluster are deployed on the same PM, thereby compromising fault tolerance. In our case, the two Sprout instances are on the same PM, just as the two Bono instances.

6.3 Optimized deployments

We now turn to our optimization framework to guide the deployment. To this end, we have to specify the resource usage of the various VMs and whether we want these capacities to be guaranteed, the number of CPU cores the VMs need and whether cores must be dedicated, and any outstanding "future" reservations. For our example, this is given by Table 3. Moreover, to guarantee a higher fault tolerance level, we declare that at most 1 Bono instance and at most 1 Sprout instance may be allocated on a PM.

We solve the ILP problem with various weights. In the first run, α was set high (consolidation is the most

VM	Core utilizations (GHz)	Memory (GB)	IOPS	PM (initial)
Sprout1	5,2	2	10	bigPM1
Sprout2	3,4	4	45	bigPM1
Bono1	3	1	0	bigPM2
Bono2	2	1	0	bigPM2
Homer	2	3	400	smallPM1
Homestead	3	4	300	smallPM2
otherVM1	1,4	1	100	bigPM1
otherVM2	1,2,1,3	4	100	smallPM2

Table 2: Resource utilization of VMs in the initial deployment

VMTypes	Cores /dedicated?	Capacity /guaranteed?	Memory /guaranteed?	IOPS /guaranteed?	Reserved
Bono	1/y	4/y	1/y	100/n	1
Sprout	2/n	6/y	4/y	100/n	0
Homer	1/n	4/n	4/y	1000/y	0
Homestead	1/n	4/n	4/y	1000/y	0
other1	2/n	4/n	4/n	100/n	0
other2	4/n	2/n	4/n	50/n	0

Table 3: Resource configuration for the VM types of the example

important), which led to a drastically decreased number of active PMs as shown in Figure 4. This deployment is nicely consolidated and also has the desired anti-affinity properties. However, also some disadvantages can be observed. There are too many VMs of the same tenant on a single node, leading to big impact if one of the machines fails. The other problem is that there is no reserve in the system for the future Bono instance (according to Table 3, the tenant specified there will be one more Bono instance in the future): since Bono instances must not be colocated, when the tenant requests the third Bono instance, a new PM must be started, which takes non-negligible time.

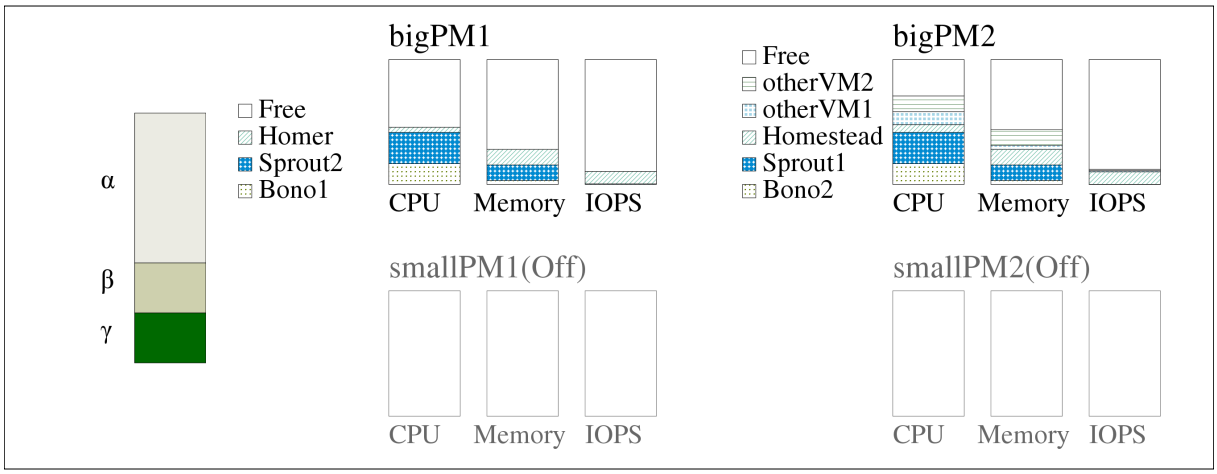

Figure 4: Allocation with high α value (consolidation is the most important)

The second run was performed with higher β, which increases the importance of future VM reservations. The resulting deployment (see Figure 5) allocates capacity for the reserved future VM as well, while still using only 3 PMs as opposed to the original 4. The reservation is visible on smallPM2 under the name of "Bono3". Of course, this allocation also obeys all the capacity and anti-affinity constraints.

The last run was configured with high γ, aiming at a better distribution of the VMs belonging to the same tenant. The solver achieves a deployment where the maximum number of VMs of the same tenant on the same PM is two (in the previous allocations, this was three), and the number of active PMs is still only three, as shown in Figure 6.

Our case study shows that changing the weights in accordance with different provider priorities changes the engineering characteristics of the deployment in the intended way. Reconciling the three, basically contradictory optimization goals and finding the weight set that best describes the intentions of a provider is a problem that we do not address here; in the future, we plan to evaluate the applicability of the standard approaches in decision theory and operations research.

6.4 Preliminary scalability assessment

In order to assess the scalability of our approach, we used a synthetic scaleup of the previously shown setup: replicating the 4 physical hosts n times, resulting in a mini-DC with $4n$ PMs and also replicating the set of VMs to be allocated n times ($n = 2, 3, 4 \ldots$). The solver was run on a workstation with Intel Core i7 processor and 16GB RAM. We specified a 30 second cutoff threshold for the solver. When an optimal solution is not found within this time limit, then solving is interrupted and the solver returns the solution with the best objective value as well as

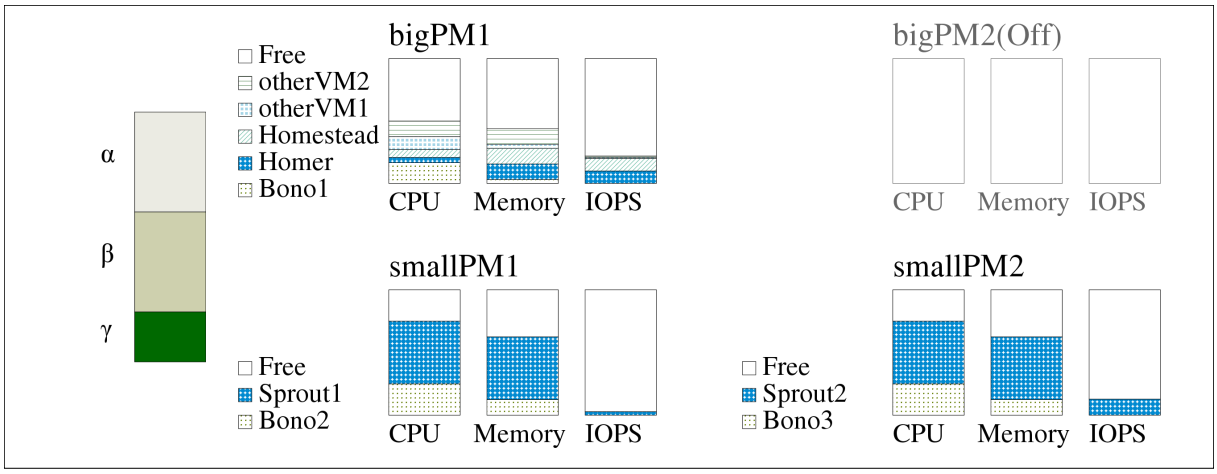

Figure 5: Allocation with higher β value (increased importance of future VM reservations)

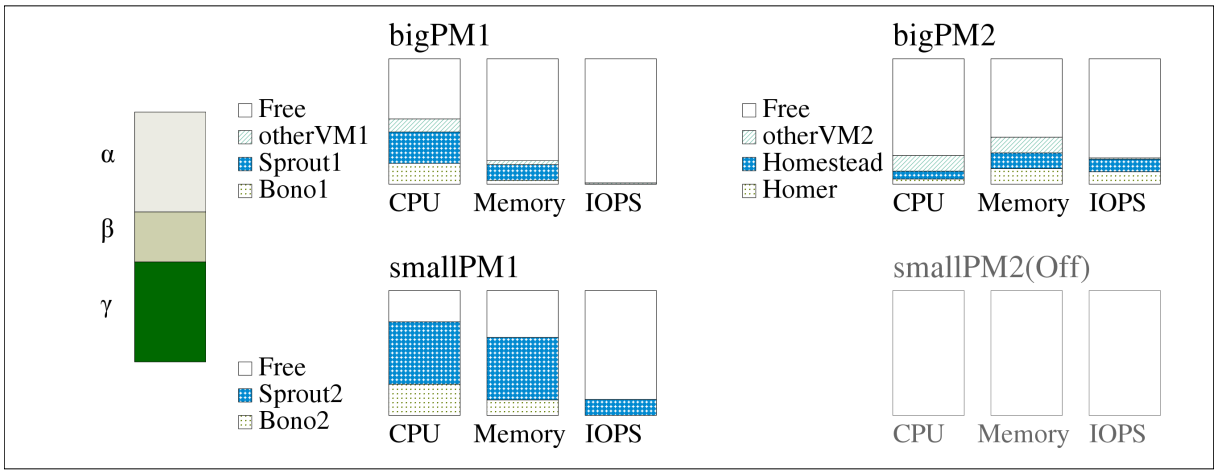

Figure 6: Allocation with higher γ value (increased importance of spreading the VMs of a tenant across PMs)

the best lower bound on the optimum that it has been able to find until that point. The solver we employ uses randomization as well as parallelization; unsurprisingly, there is significant variance in the runtime (and enforced solving interruption) from approximately 100 physical hosts. Therefore, we chose to show here the results for a single solving campaign using a single thread and the same seed value.

Figure 7 shows the best objective values found and the reported bounds; Figure 8 shows the solver runtimes. This scaling profile is representative in the sense that runtime – as expected – increases exponentially; however, we can still get near-optimal results in half a minute even for approximately 150 physical hosts. Arguably, our base scenario as well as the way we scale it up are somewhat artificial and further evaluation is needed – still, this experiment already shows that the approach is feasible at least for a few dozens of servers when fast decisions have to be made; for longer term DC reoptimization and decision precomputation (when it is reasonable to allow runtimes of tens of minutes), this scale is at least one order of magnitude larger.

It was clear from the start of this work that pure ILP modeling and solving using off-the-shelf tools is not a feasible approach for DC-scale problems. Our main focus was a first attempt at laying the modeling groundwork; there are numerous techniques that have the promise of enabling efficient decision-making at DC-scale. The modification of various existing heuristics for DC allocation optimization is just one option; we also plan to evaluate the applicability of incremental solving techniques and hierarchical approaches. That said, it is important to note that even with off-the-shelf solvers, we are already in the full single-rack size domain – making our results already directly applicable inside the typical "building blocks" of DCs. Intra-rack allocation is in itself an important problem. As the communication between the hosts is here the fastest (and the overall cost to move data between hosts the lowest), many distributed applications do prefer all tightly coupled components to be hosted in the same rack (possibly with additional components at the macro scale; e.g. loosely coupled instantiations of these groups "elsewhere"). Our approach can be applied directly for placement optimization in such cases.

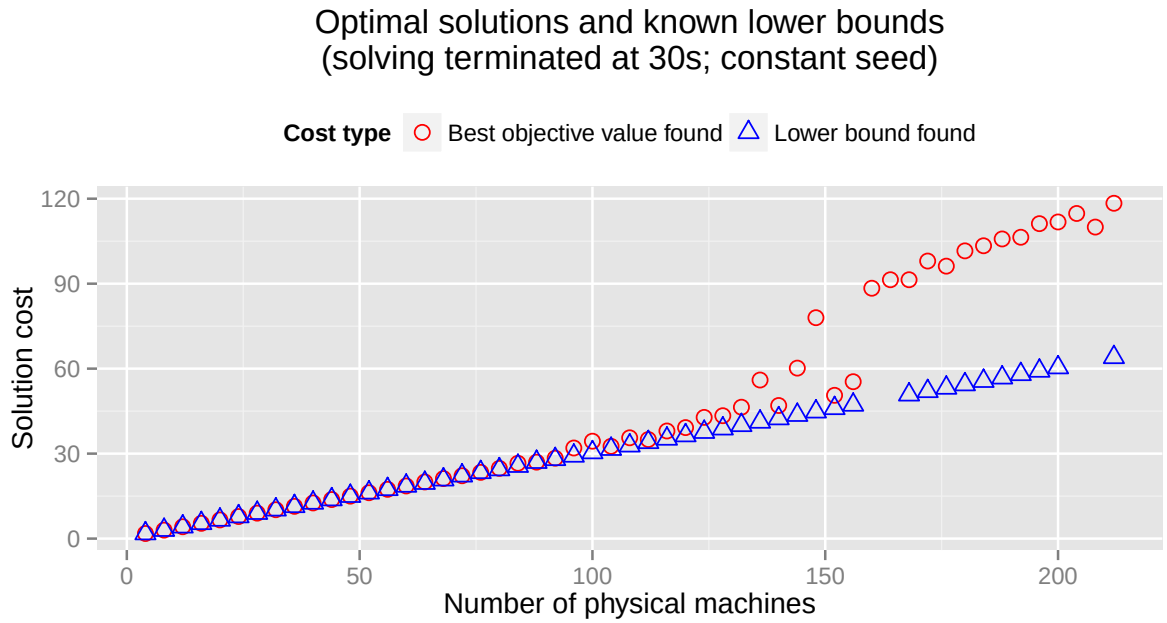

Figure 7: Best objective value and best lower bound found for case study scaling

7 Conclusions

In this paper, we have extended the classic data center allocation optimization problem for critical tenant applications that need guarantees on the resource capacities they wish to consume. We identified a set of representative, user-issuable constraints and new optimization objectives, and established a mathematical and corresponding integer programming formulation that combines tenant and provider goals in a joint optimization problem. Using a typical NFV application, we have shown the viability of the approach and presented initial scalability results. Further work will focus on establishing solution approaches for the modeling framework that scale to massive amounts of hosts as well as modeling further critical applications to increase practical applicability.

Acknowledgements

The work of all three authors was partially supported by 2015 research scholarships of the Pro Progressio Foundation in the topic of performance analysis of cloud-based systems. The work of Z. Á. Mann was partially supported by the Hungarian Scientific Research Fund (Grant Nr. OTKA 108947).

References

[1] 3GPP. IP Multimedia Subsystem (IMS); Stage 2. `http://www.3gpp.org/DynaReport/23228.htm`, 2015.

[2] Ehsan Ahvar, Shohreh Ahvar, Noel Crespi, Joaquin Garcia-Alfaro, and Zoltán Á. Mann. NACER: a network-aware cost-efficient resource allocation method for processing-intensive tasks in distributed clouds. In *Proceedings of the 14th IEEE International Symposium on Network Computing and Applications*, pages 90–97, 2015.

[3] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[4] Dávid Bartók and Zoltán Á. Mann. A branch-and-bound approach to virtual machine placement. In *Proceedings of the 3rd HPI Cloud Symposium "Operating the Cloud"*, pages 49–63, 2015.

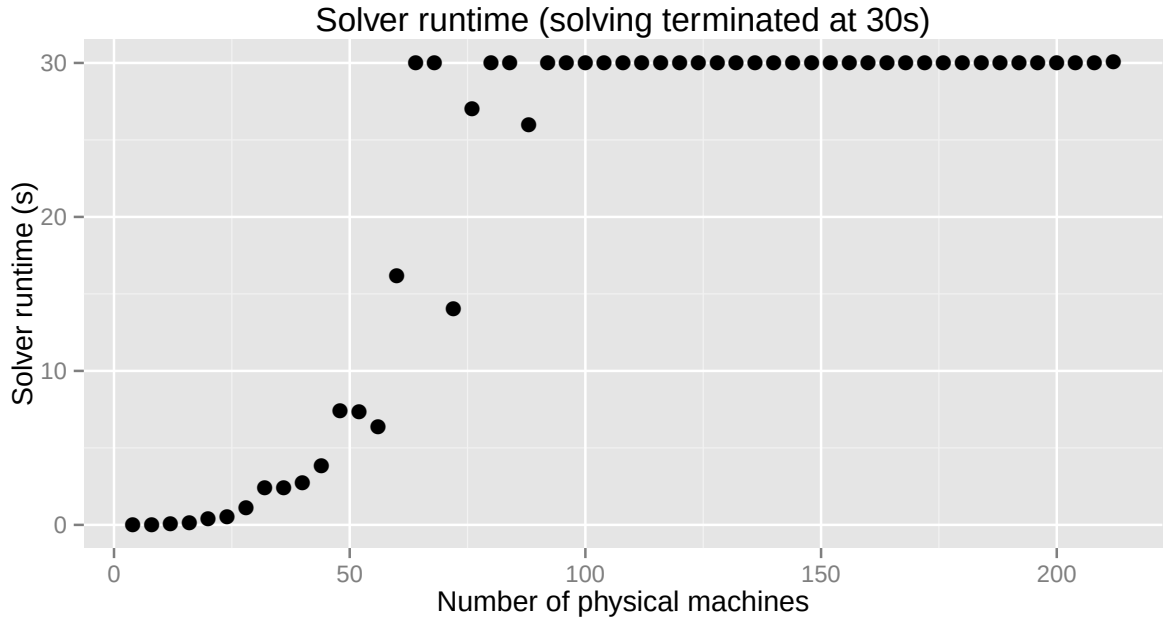

Figure 8: Solver runtime for case study scaling (with 30s cutoff)

[5] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28:755–768, 2012.

[6] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[7] Ofer Biran, Antonio Corradi, Mario Fanelli, Luca Foschini, Alexander Nus, Danny Raz, and Ezra Silvera. A stable network-aware VM placement for cloud systems. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, pages 498–506. IEEE Computer Society, 2012.

[8] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing SLA violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, 2007.

[9] David Breitgand and Amir Epstein. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In *Proceedings of IEEE Infocom 2012*, pages 2861–2865, 2012.

[10] Mohammed Rashid Chowdhury, Mohammad Raihan Mahmud, and Rashedur M. Rahman. Study and performance analysis of various VM placement strategies. In *16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2015.

[11] CSMIC. SMI Framework version 2.1. `http://csmic.org/wp-content/uploads/2014/07/SMI_Overview_TwoPointOne1.pdf`, 2013.

[12] Sevil Dräxler, Holger Karl, and Zoltán Á. Mann. Joint optimization of scaling and placement of virtual network services. In *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2017)*, 2017.

[13] ETSI. NFV specifications. `http://www.etsi.org/technologies-clusters/technologies/nfv`, 2013.

[14] ETSI. Network Functions Virtualisation (NFV); Service Quality Metrics - ETSI GS NFV-INF 010. `http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/010/01.01.01_60/gs_NFV-INF010v010101p.pdf`, 2014.

[15] European Telecommunications Standards Institute. Network functions virtualisation – introductory white paper. https://portal.etsi.org/NFV/NFV_White_Paper.pdf, 2012.

[16] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79:1230–1242, 2013.

[17] Rahul Ghosh and Vijay K. Naik. Biting off safely more than you can chew: Predictive analytics for resource over-commit in IaaS cloud. In *5th International Conference on Cloud Computing*, pages 25–32. IEEE, 2012.

[18] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Guillaume Belrose, Tom Turicchi, and Alfons Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *IEEE International Conference on Dependable Systems and Networks*, pages 326–335, 2008.

[19] Marco Guazzone, Cosimo Anglano, and Massimo Canonico. Exploiting VM migration for the automated power and performance management of green cloud computing systems. In *First International Workshop on Energy Efficient Data Centers (E2DC 2012)*, pages 81–92. Springer, 2012.

[20] Brian Guenter, Navendu Jain, and Charles Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *Proceedings of IEEE INFOCOM*, pages 1332–1340. IEEE, 2011.

[21] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015.

[22] Sijin He, Li Guo, Moustafa Ghanem, and Yike Guo. Improving resource utilisation in the cloud environment using multivariate probabilistic models. In *IEEE 5th International Conference on Cloud Computing*, pages 574–581, 2012.

[23] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 104–113, 2011.

[24] Gueyoung Jung, Matti A. Hiltunen, Kaustubh R. Joshi, Richard D. Schlichting, and Calton Pu. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *IEEE 30th International Conference on Distributed Computing Systems (ICDCS)*, pages 62–73, 2010.

[25] Shin-gyu Kim, Hyeonsang Eom, and Heon Y. Yeom. Virtual machine consolidation based on interference modeling. *The Journal of Supercomputing*, 66(3):1489–1506, 2013.

[26] Imre Kocsis, András Pataricza, Zoltán Micskei, András Kövi, and Zsolt Kocsis. Analytics of resource transients in cloud-based applications. *International Journal of Cloud Computing*, 2(2-3):191–212, 2013.

[27] Daniel Guimaraes do Lago, Edmundo R. M. Madeira, and Luiz Fernando Bittencourt. Power-aware virtual machine scheduling on clouds using active cooling control and DVFS. In *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, 2011.

[28] Wubin Li, Johan Tordsson, and Erik Elmroth. Virtual machine placement for predictable and time-constrained peak loads. In *Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON 2011)*, pages 120–134. Springer, 2011.

[29] Zheng Li, Liam OBrien, Rainbow Cai, and He Zhang. Towards a Taxonomy of Performance Evaluation of Commercial Cloud Services. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 344–351. IEEE, June 2012.

[30] Zoltán Ádám Mann. Allocation of virtual machines in cloud data centers – a survey of problem models and optimization algorithms. *ACM Computing Surveys*, 48(1), 2015.

[31] Zoltán Ádám Mann. Approximability of virtual machine allocation: much harder than bin packing. In *Proceedings of the 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, pages 21–30, 2015.

[32] Zoltán Ádám Mann. Modeling the virtual machine allocation problem. In *Proceedings of the International Conference on Mathematical Methods, Mathematical Models and Simulation in Science and Engineering*, pages 102–106, 2015.

[33] NIST. The NIST Definition of Cloud Computing - SP 800-145. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011.

[34] Project Clearwater. Online documentation for Project Clearwater. http://clearwater.readthedocs.org/en/stable/index.html, 2015.

[35] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symposium on Cloud Computing (SoCC)*, 2012.

[36] Bruno Cesar Ribas, Rubens Massayuki Suguimoto, Razer A. N. R. Montano, Fabiano Silva, Luis de Bona, and Marcos A. Castilho. On modelling virtual machine consolidation to pseudo-boolean constraints. In *13th Ibero-American Conference on AI*, pages 361–370, 2012.

[37] Ivan Rodero, Hariharasudhan Viswanathan, Eun Kyung Lee, Marc Gamell, Dario Pompili, and Manish Parashar. Energy-efficient thermal-aware autonomic management of virtualized HPC cloud infrastructure. *Journal of Grid Computing*, 10(3):447–473, 2012.

[38] Lei Shi, John Furlong, and Runxin Wang. Empirical evaluation of vector bin packing algorithms for energy efficient data centers. In *IEEE Symposium on Computers and Communications (ISCC)*, pages 9–15, 2013.

[39] Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. Adaptive resource provisioning for the cloud using online bin packing. *IEEE Transactions on Computers*, 63(11):2647–2660, 2014.

[40] Luis Tomás and Johan Tordsson. An autonomic approach to risk-aware data center overbooking. *IEEE Transactions on Cloud Computing*, 2(3):292–305, 2014.

[41] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 USENIX Annual Technical Conference*, pages 355–368, 2009.

[42] Joe Weinman. *Cloudonomics: The Business Value of Cloud Computing*. John Wiley & Sons, 2012.

[43] Xiaoyun Zhu, Donald Young, Brian J. Watson, Zhikui Wang, Jerry Rolia, Sharad Singhal, Bret McKee, Chris Hyser, Daniel Gmach, Robert Gardner, Tom Christian, and Ludmila Cherkasova. 1000 islands: an integrated approach to resource management for virtualized data centers. *Cluster Computing*, 12(1):45–57, 2009.